Design methods: SSA/SD[†]

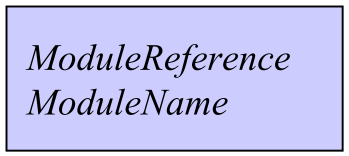
- Stands for: *Structured Systems Analysis / Structured Design*.
- Primary applicable notations:
  - DFDs
  - Structure Charts
- Secondary notations:
  - ERDs
  - Pseudo-code
  - *Data Dictionary*
- Overall objective: *Derive "white box" structure chart.*

[†] Material from text by Budgen and from "Software Engineering: A Practitioner's Approach (4th edition)", by Roger Pressman.

Effective modular design

Module: a grouping of related routines or data.

- Diagrammatic convention:



- Modularization criteria:
  - *coupling*: The degree of interconnection between modules.
  - *cohesion*: The strength of relationship between elements of a particular module; the “single mindedness of purpose” of the module.

Cohesion and *coupling* are at odds

- Improving one tends to worsen the other.

$$\begin{tikzcd}
\text{(\textit{more single mindedness of purpose})} \arrow[d] & \text{(\textit{less communication})} \arrow[d] \\
\text{(\textit{more modules})} \arrow[d] & \text{(\textit{fewer modules})} \arrow[d] \\
\text{(\textit{more communication})} & \text{(\textit{less single mindedness of purpose})} \\
\text{(a)} & \text{(b)}
\end{tikzcd}$$

Degrees of *coupling*

Strongest —— (*more desirable*) $\longrightarrow$ weakest:

1. **Content coupling**: one module can change the local data or control of another. Usually not possible with high level languages.
2. **Common coupling**: a single shared global data structure.
3. **Control coupling**: indirect execution control of one module by another (e.g., by passing control information in parameters).
4. **Stamp coupling**: multiple shared global data structures; fewer modules share a particular subset of the global data.
5. **Data coupling**: All communication of data is via parameters.

Degrees of *cohesion*

Weakest —— (*more desirable*) ⟶ strongest:

1. **Coincidental cohesion**: No apparent relationship.
2. **Logical cohesion**: Some minimal relationship (e.g., all I/O routines).
3. **Temporal cohesion**: Some minimal relationship and all parts execute at the same time (e.g., all initialization code).
4. **Communication cohesion**: Some minimal relationship and all parts execute at the same time on the same data.
5. **Sequential cohesion**: The elements are in a sequential pipe-and-filter sequence.

Degrees of *cohesion* (cont'd)

Weakest —— (*more desirable*) $\longrightarrow$ strongest:

6. **Functional cohesion**: All elements are related to the performance of a single function (e.g., all procedure that computer a square root)..

7. **Informational cohesion**: A module corresponds to an abstract data type.

SSA/SD Process (from text and Pressman)

The first three steps.

1. Construct an initial DFD for each major component to provide a top-level description of the problem (the *context diagrams*).

2. Review and refine DFDs for the major components until a sufficient degree of cohesion is achieved for processes; one elaborates the context diagrams into a layered hierarchy of DFDs, supported by a data dictionary.

3. Determine whether each DFD has *transformational* or *transactional* flow characteristics.

The remaining steps depend on the outcome of step 3.

Flow types

Transformational Flow

Data "continuously" moves through a collection of incoming flow processes, transform center processes, and finally outgoing flow processes.

Transactional Flow

Data "continuously" moves through a collection of incoming flow processes, reaches a particular transaction center process, and then follows one of a number of actions paths. Each action path is again a collection of processes.

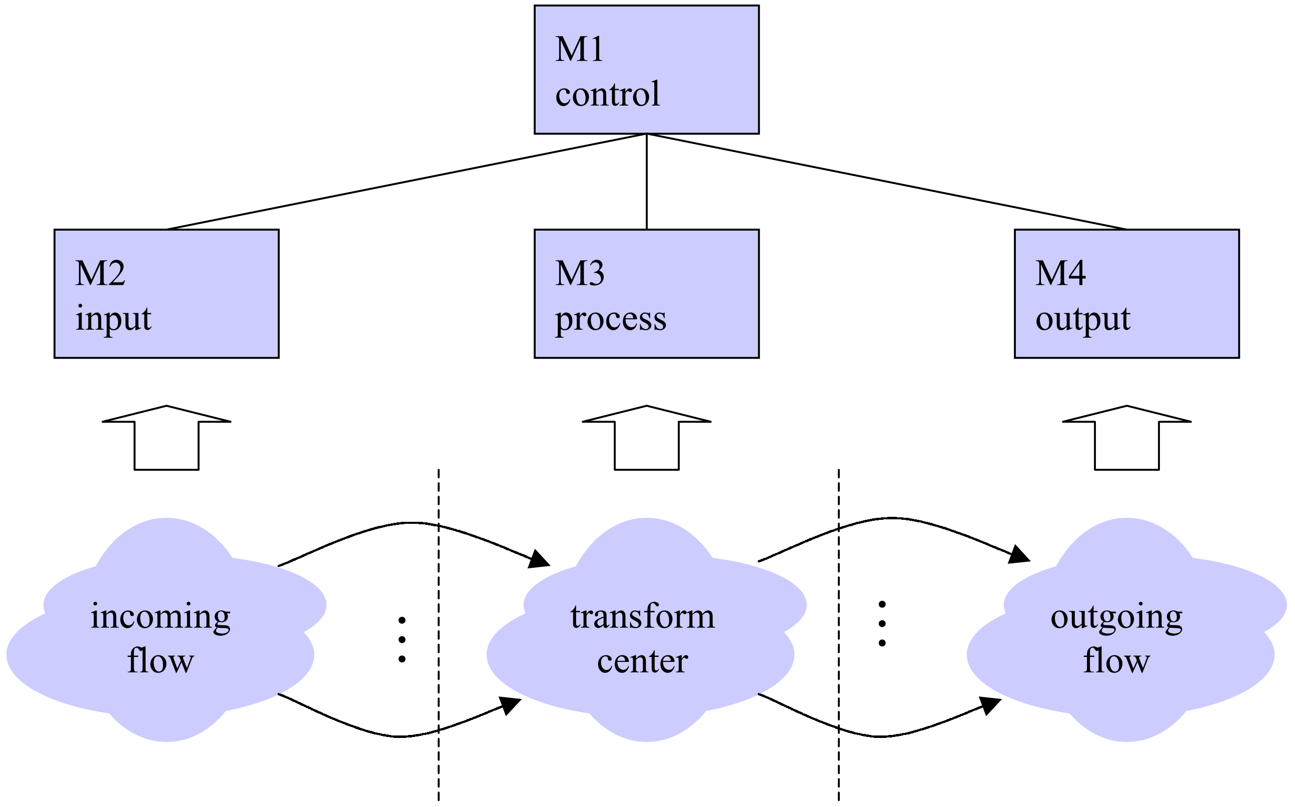
SSA/SD Process (cont’d)

Transform mapping detail.

4. Isolate the transform center by specifying incoming and outgoing flow boundaries.
5. Perform “1st –level” factoring for transformational flow (see next slide). Factoring results in a program structure in which top-level modules perform decision making and low-level modules perform, input, computation and output. (Mid-level modules can perform both.)
6. Perform “2nd-level” factoring: two or more processes become a single module; one process becomes two or more modules.
7. Refine the first iteration program structure using design heuristics for improved software quality.

SSA/SD Process (cont’d)

“1st –level” factoring in transformational flow.



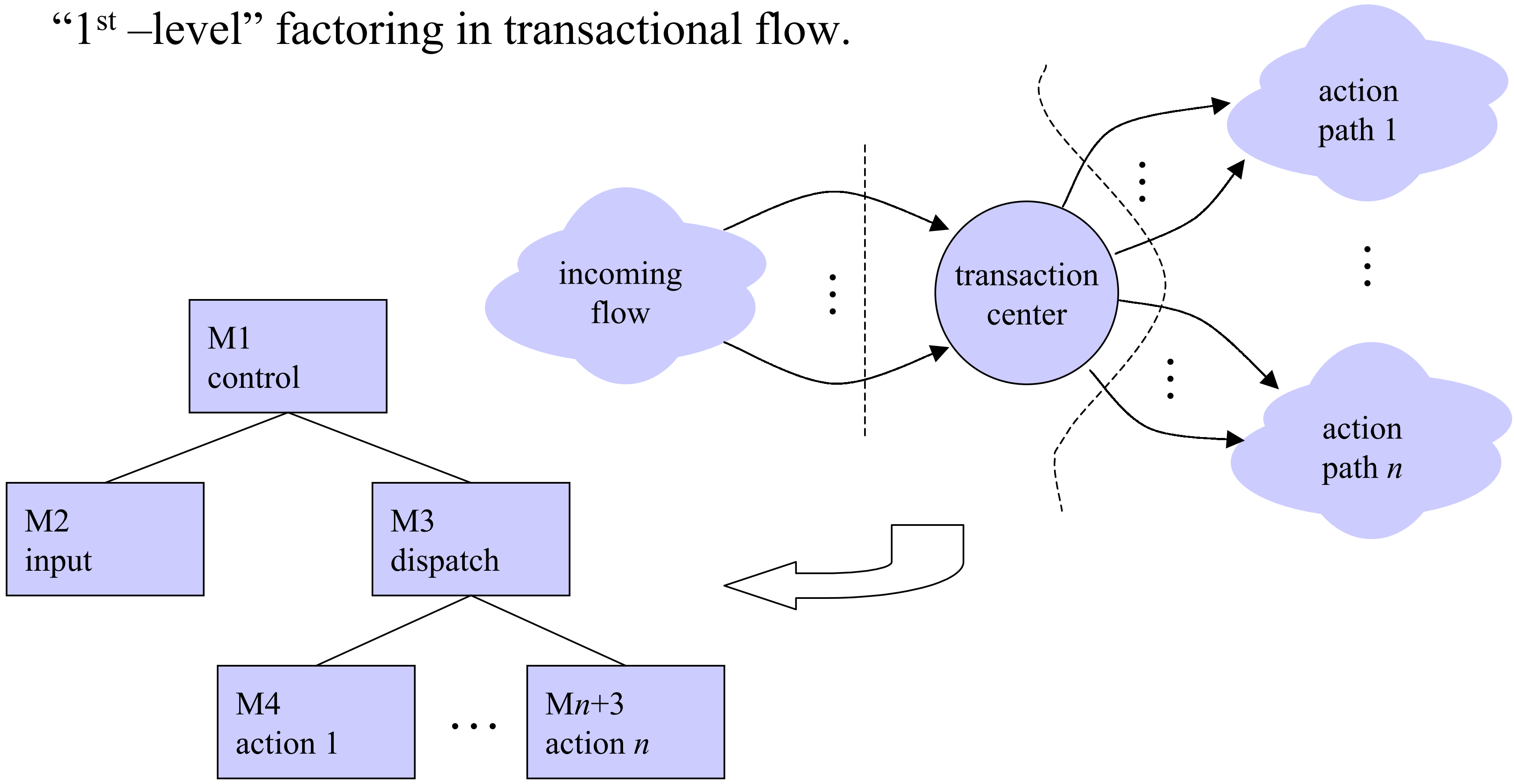
SSA/SD Process (cont’d)

Transaction mapping detail.

4. Identify the transaction center, and the flow characteristics along each of the action paths.
5. Perform “1st –level” factoring for transactional flow (see next slide); map the DFD to a program structure amenable to transaction processing.
6. Factor and refine the transaction structure and the structure of each action path.
7. Refine the first iteration program structure using design heuristics for improved software quality.

SSA/SD Process (cont'd)

"1st –level" factoring in transactional flow.



Design heuristics for effective modularity

- Reevaluate “first iteration” (employ *iterative design*).
- Minimize high *fan-out*; strive for *fan-in* as depth increases.

Scope of effect of a module: any module that contains code that is executed based on the outcome of a decision within the module.

Scope of control of a module: that module plus all modules that are subordinate to it in its associated structure chart.

- Keep *scope of effect* within *scope of control*.
- Evaluate module interfaces to reduce complexity and redundancy and improve consistency.

Design heuristics for effective modularity (cont'd)

- Define modules with transparent functionality, but avoid modules that are overly restrictive (e.g., impose size or option restrictions that seem arbitrary).

- Strive for "controlled entry" modules, avoiding "pathological cases".

- Create software components based on design constraints and portability requirements.

- Evaluate module interfaces to reduce complexity and redundancy and improve consistency.

Design postprocessing

After structure charts have been developed and refined, the following tasks must be completed.

- A processing narrative must be developed for each module.
- An interface description is provided for each module.
- Local and global data structures are refined or designed.
- All design restrictions and limitations are noted.
- A design review is conducted.
- “Optimization” is considered (if required and justified).

Case study: the *SafeHome* software system

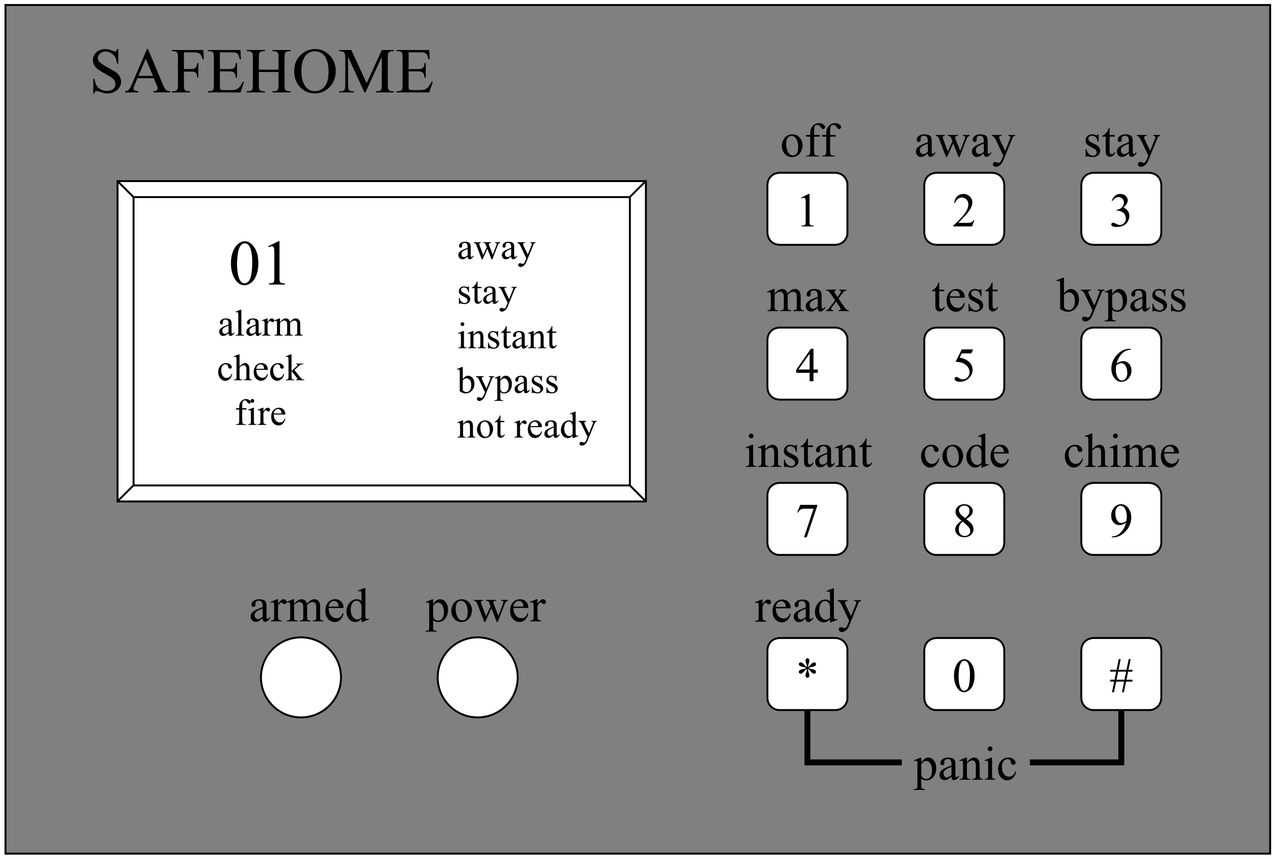
SafeHome software enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through a keypad and function keys contained in the SafeHome control panel shown below.

During installation, the SafeHome control panel is used to “program” and configure the system. Each sensor is assigned a number and type, a master password is programmed for arming and disarming the system, and telephone number(s) are input for dialing when a sensor event occurs.

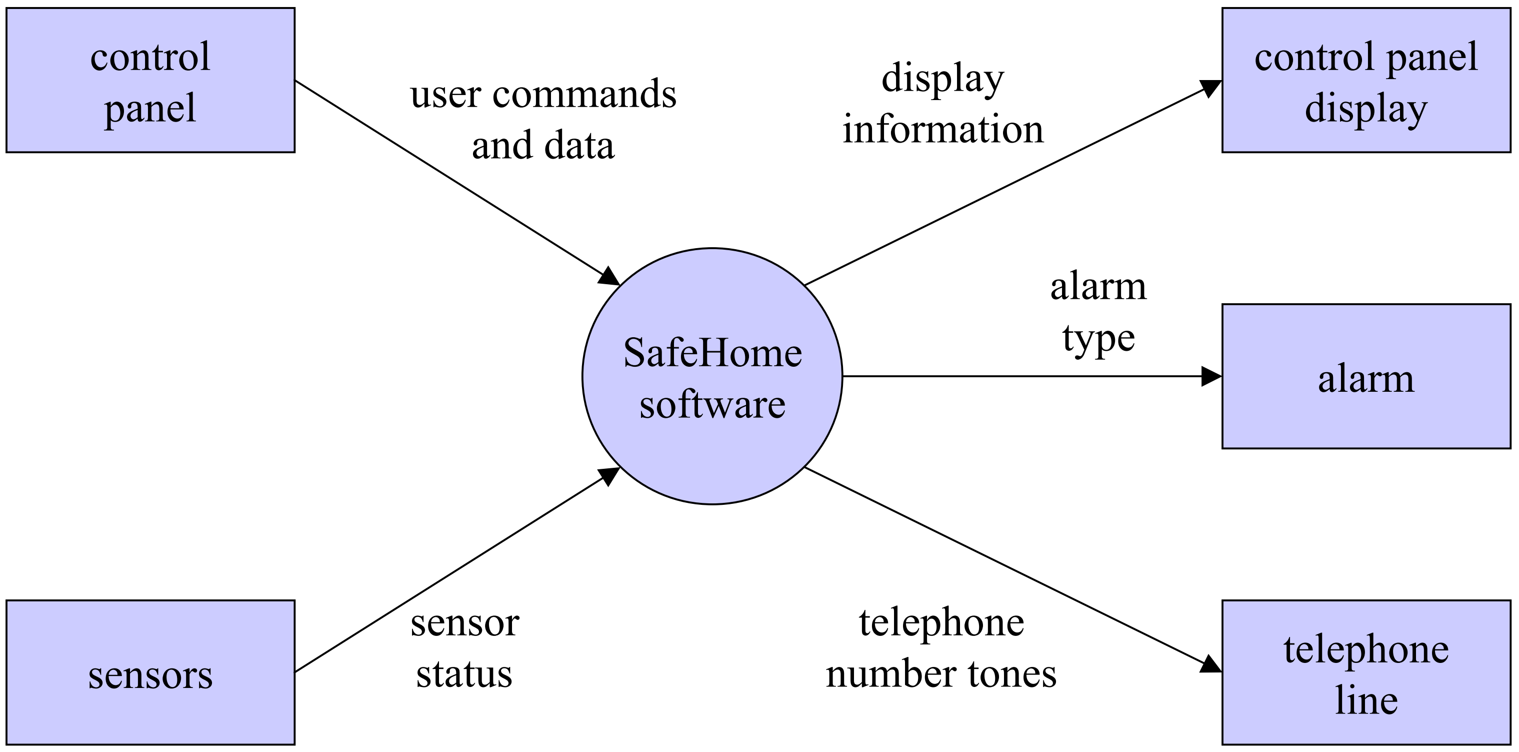
When a sensor event is recognized, the software invokes an audible alarm attached to the system. After a delay time that is specified by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, provides information about the location, reporting the nature of the event that has been detected. The number will be redialed every 20 seconds until telephone connection is obtained.

All interaction with SafeHome is managed by a user-interaction subsystem that reads input provided through the keypad and function keys, displays prompting messages and system status on the LCD display. Keyboard interaction takes the following form …

Case study (cont'd): the *SafeHome* control panel



E.g.: Level 0 for *SafeHome*



Creating a level 1 DFD

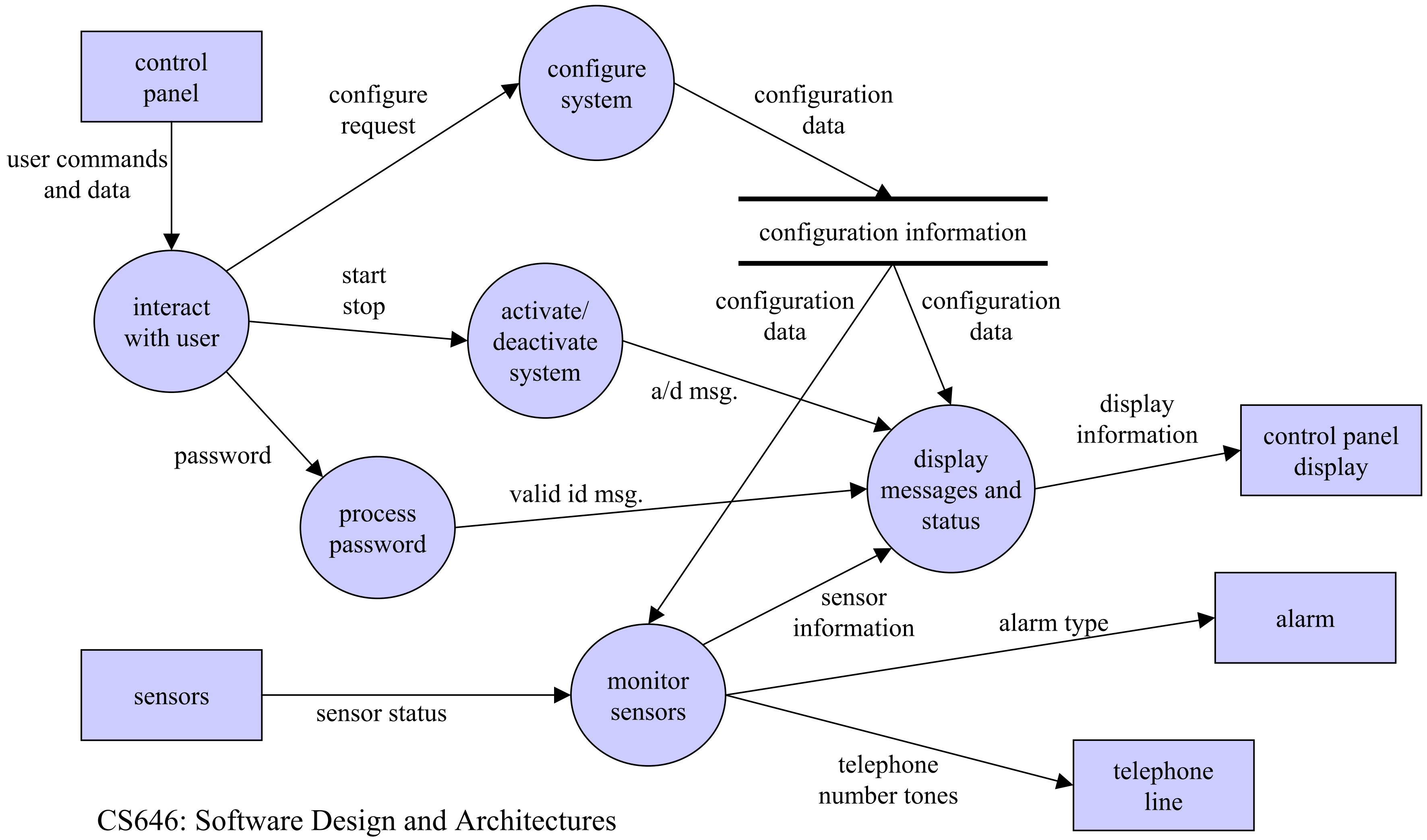
SafeHome software *enables* the homeowner to *configure* the security system when it is *installed*, *monitors* all sensors *connected* to the security system, and *interacts* with the homeowner through a keypad and function keys *contained* in the SafeHome control panel shown below.

During installation, the SafeHome control panel is *used* to “*program*” and *configure* the system. Each sensor is *assigned* a number and type, a master password is programmed for *arming* and *disarming* the system, and *telephone number(s)* are *input* for *dialing* when a sensor event occurs.

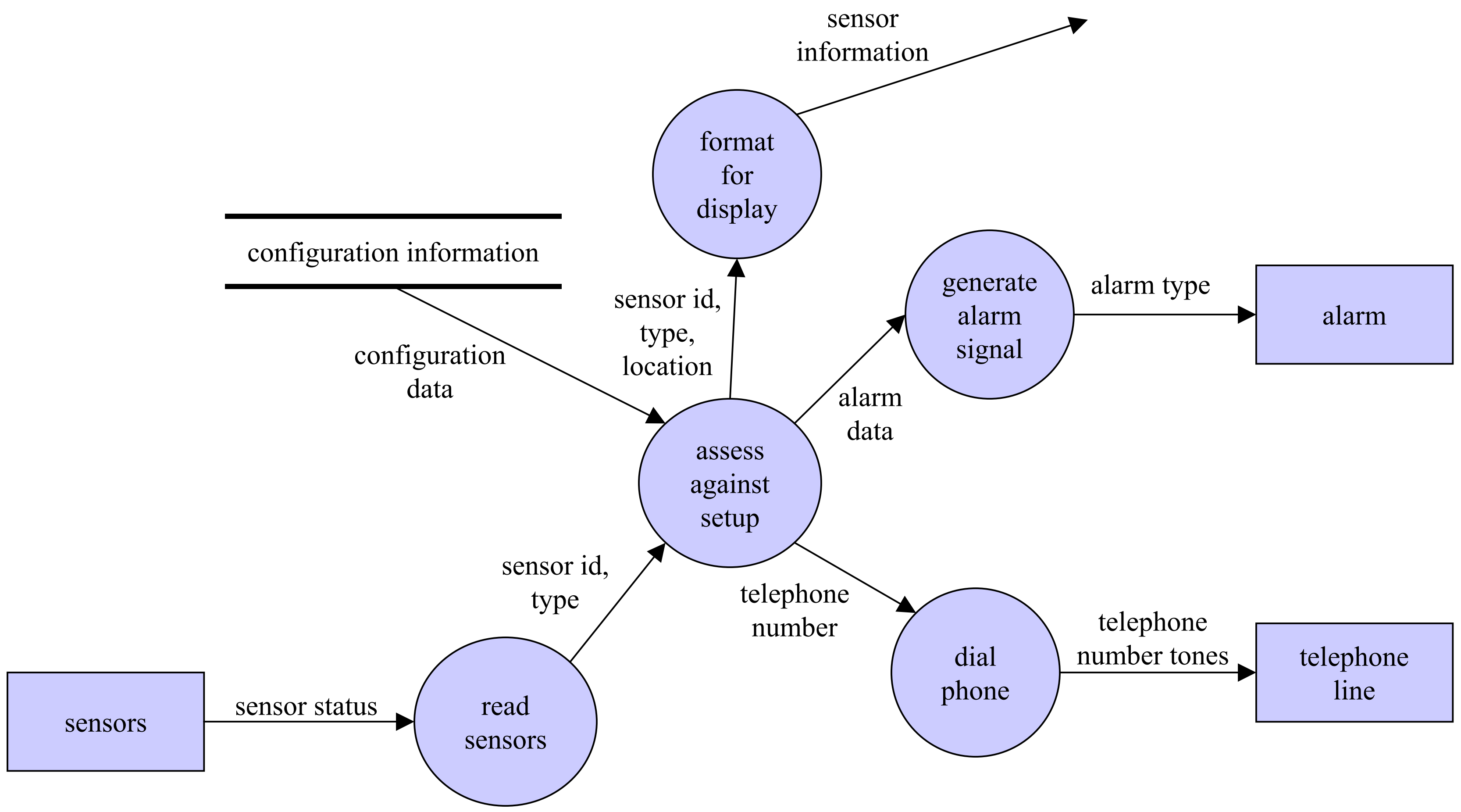
When a sensor event is *recognized*, the software *invokes* an audible alarm attached to the system. After a delay time that is *specified* by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, *provides* information about the location, *reporting* the nature of the event that has been detected. The number will be *redialed* every 20 seconds until telephone connection is *obtained*.

All interaction with SafeHome is *managed* by a user-interaction subsystem that *reads* input provided through the keypad and function keys, *displays* prompting messages and system status on the LCD display. Keyboard interaction takes the following form …

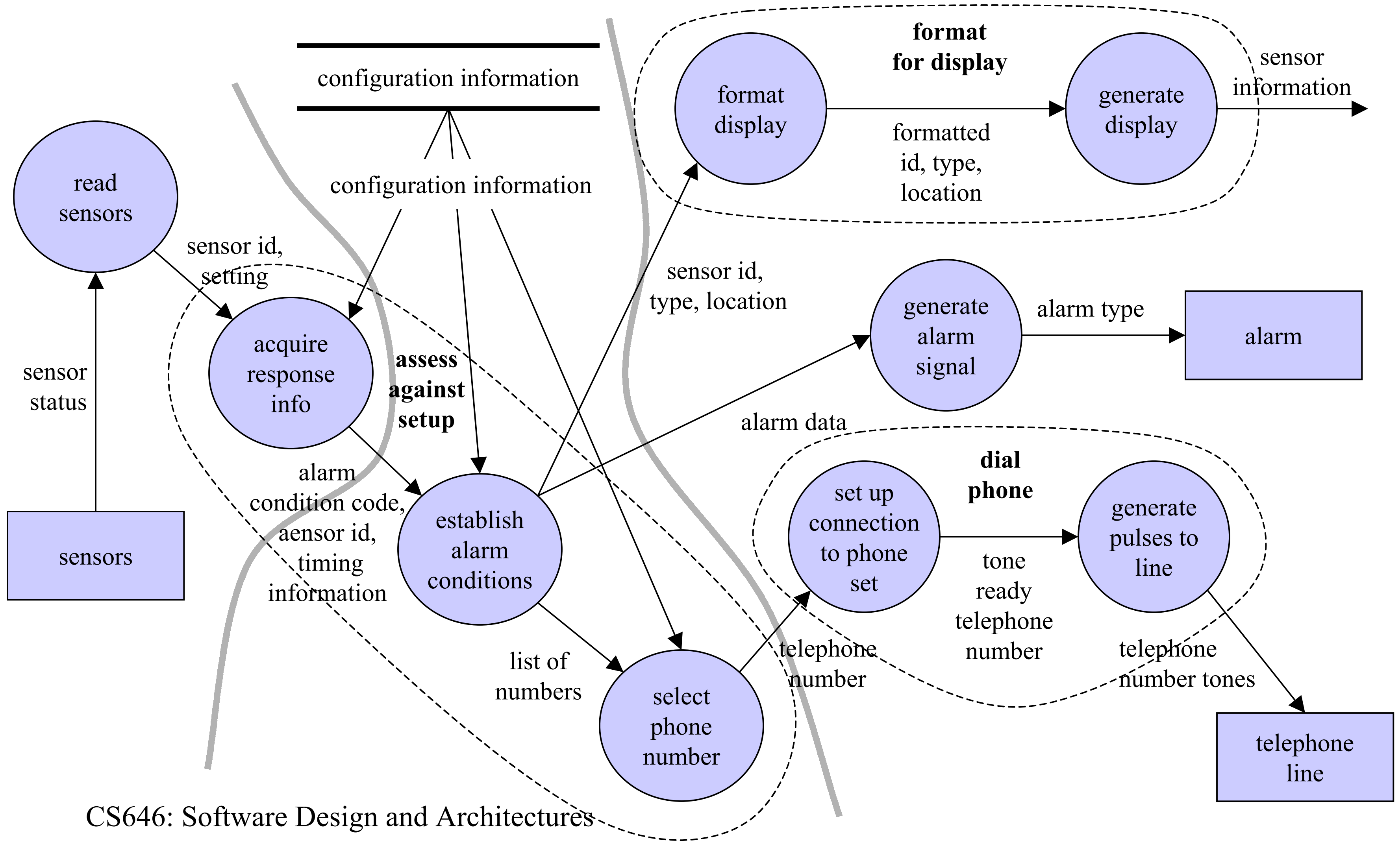
E.g. (cont’d): Level 1 (*SafeHome software*)



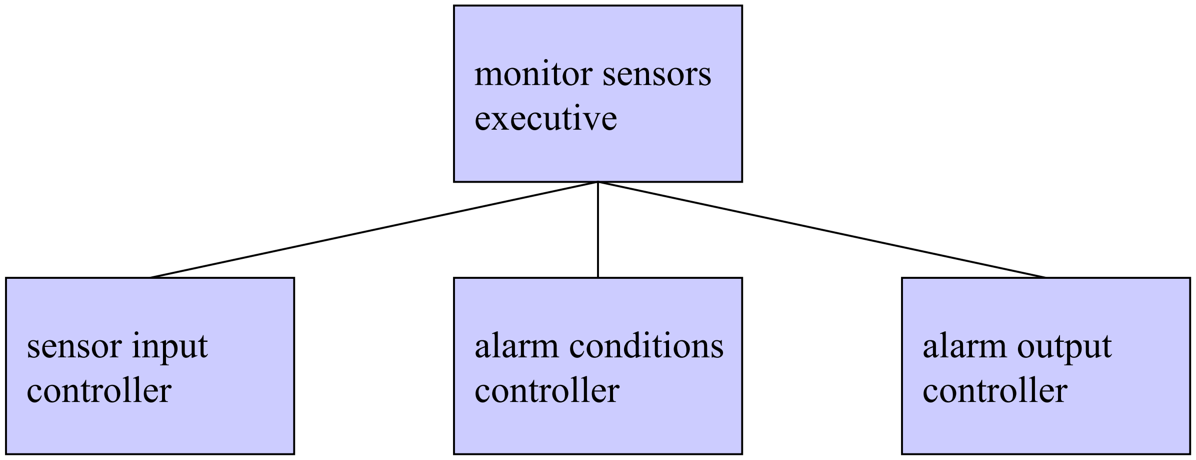
E.g. (cont'd): Level 2 (*monitor sensors*)



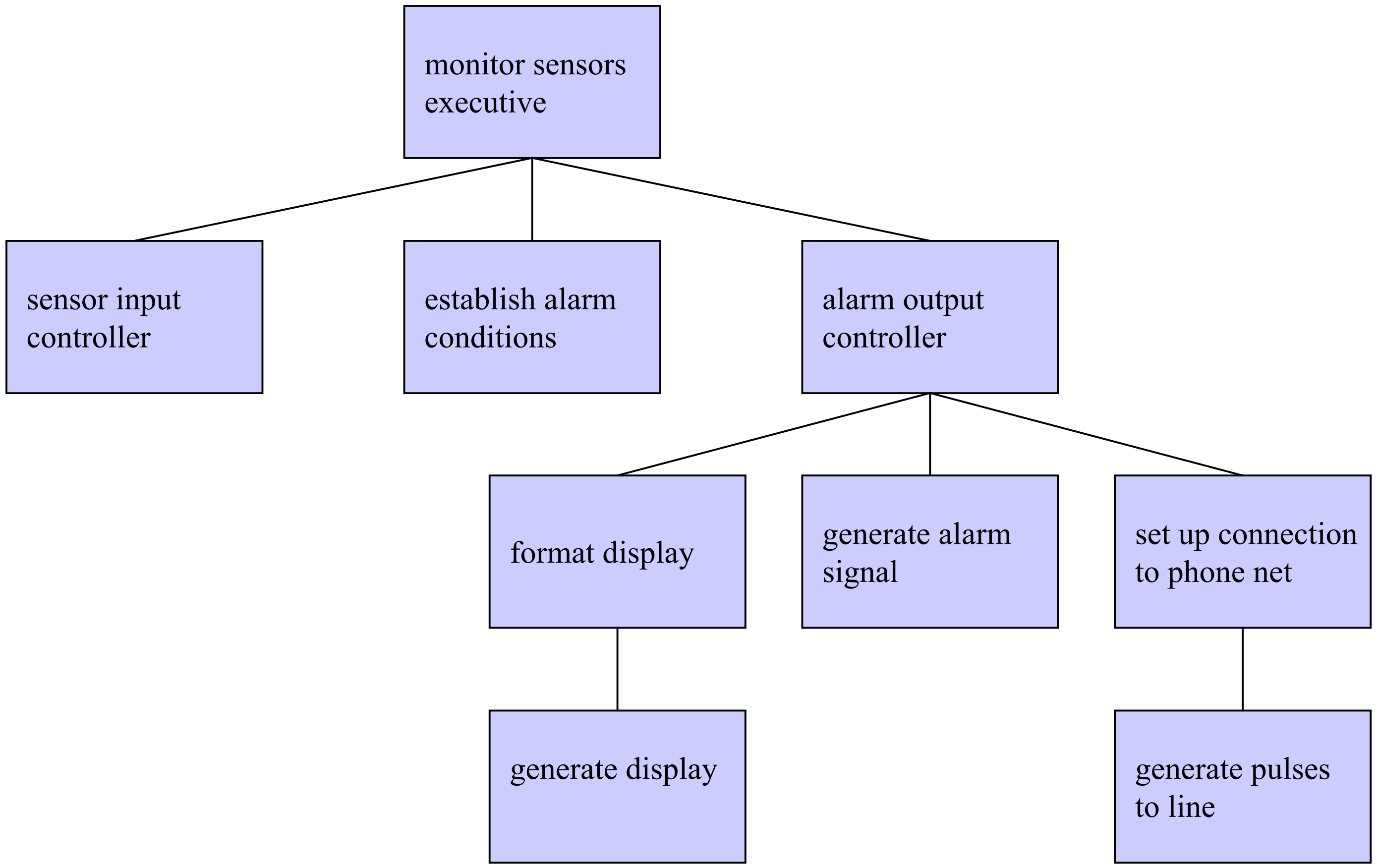
E.g. (cont'd): Level 3 (*monitor sensors*)



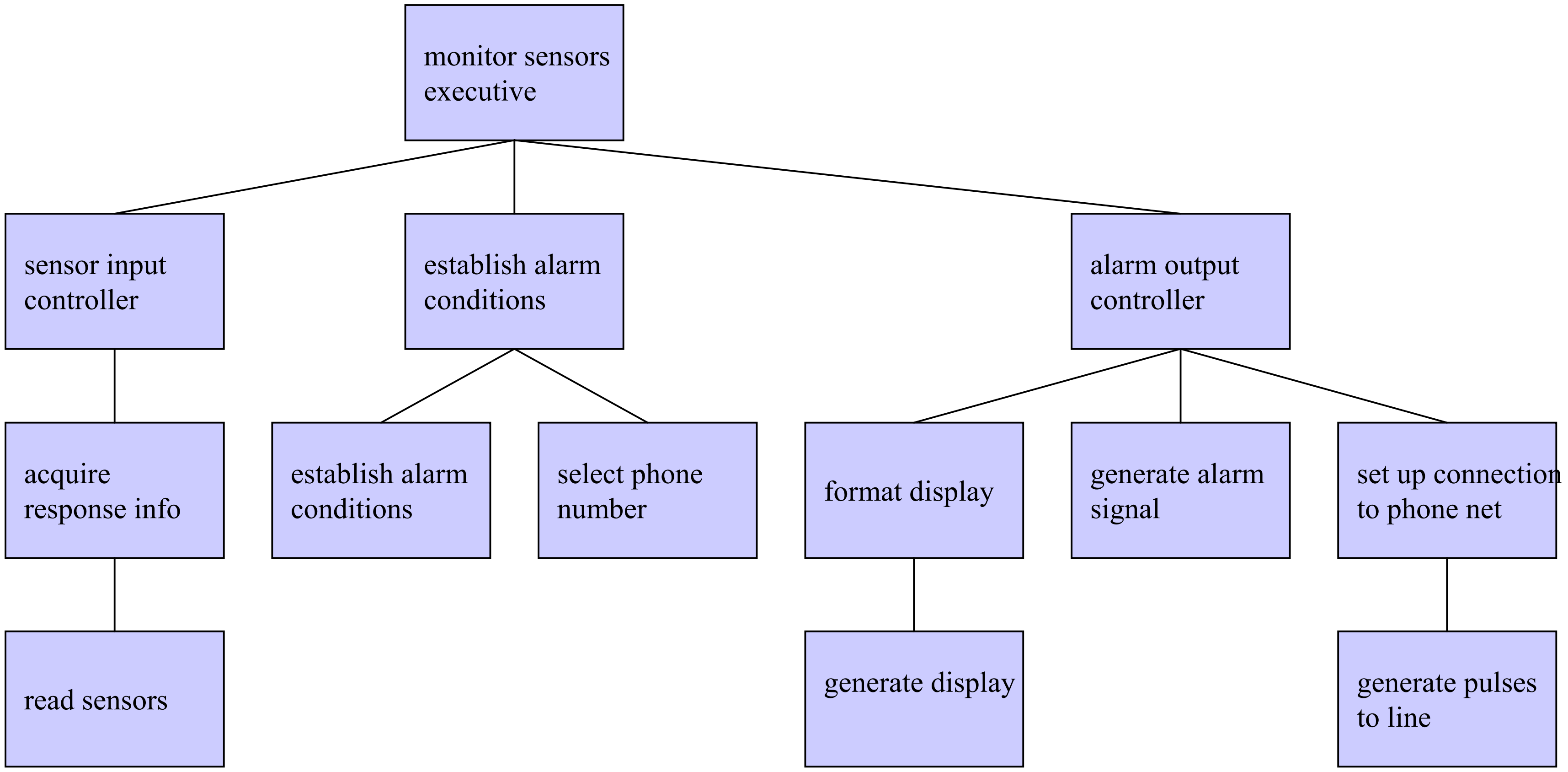
E.g. (cont'd) 1st-level factoring (*monitor sensors*)



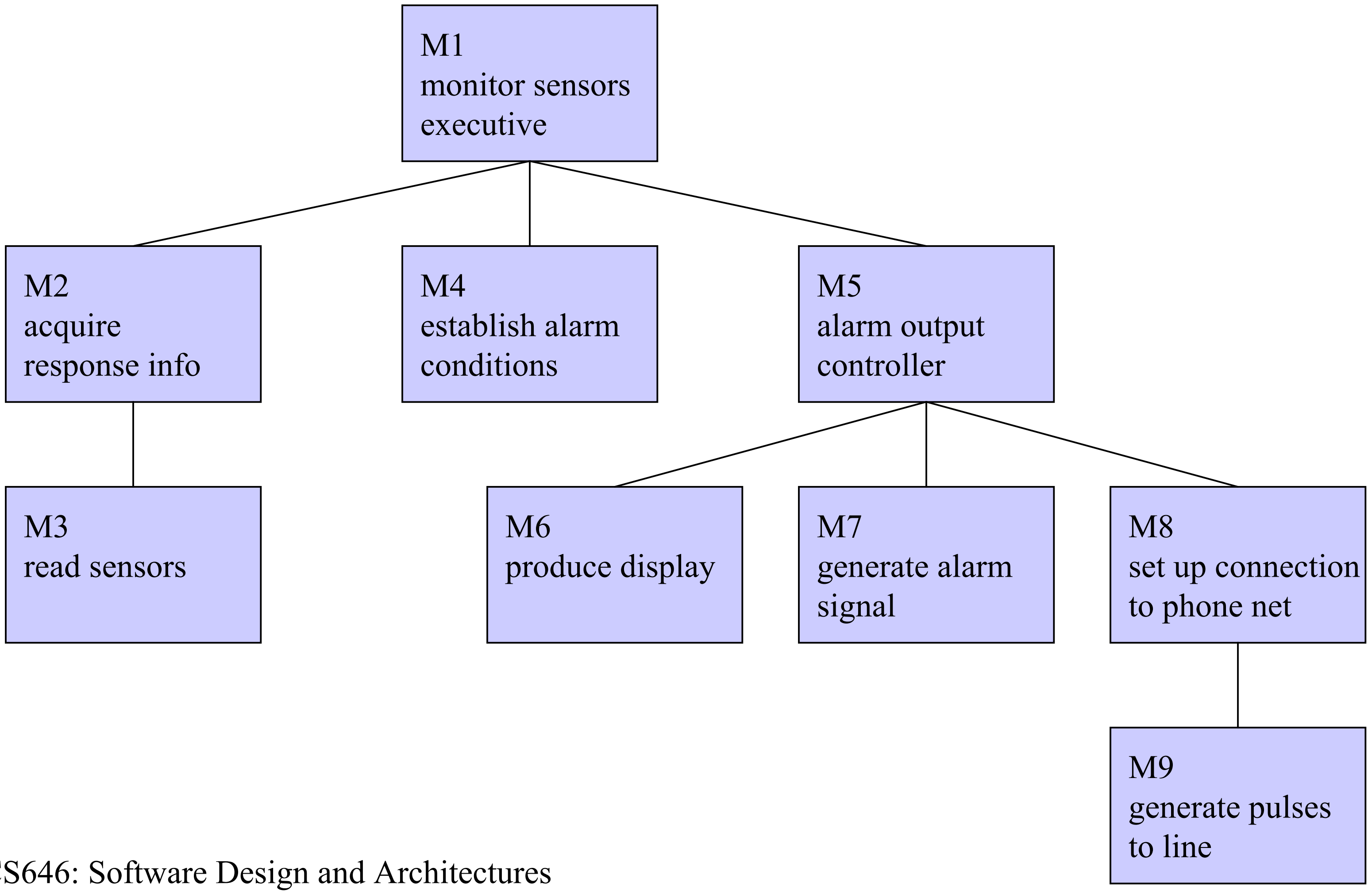
E.g. (cont'd) 1st-level factoring (*monitor sensors*)



E.g. (cont'd) 1st-cut program structure (*monitor sensors*)

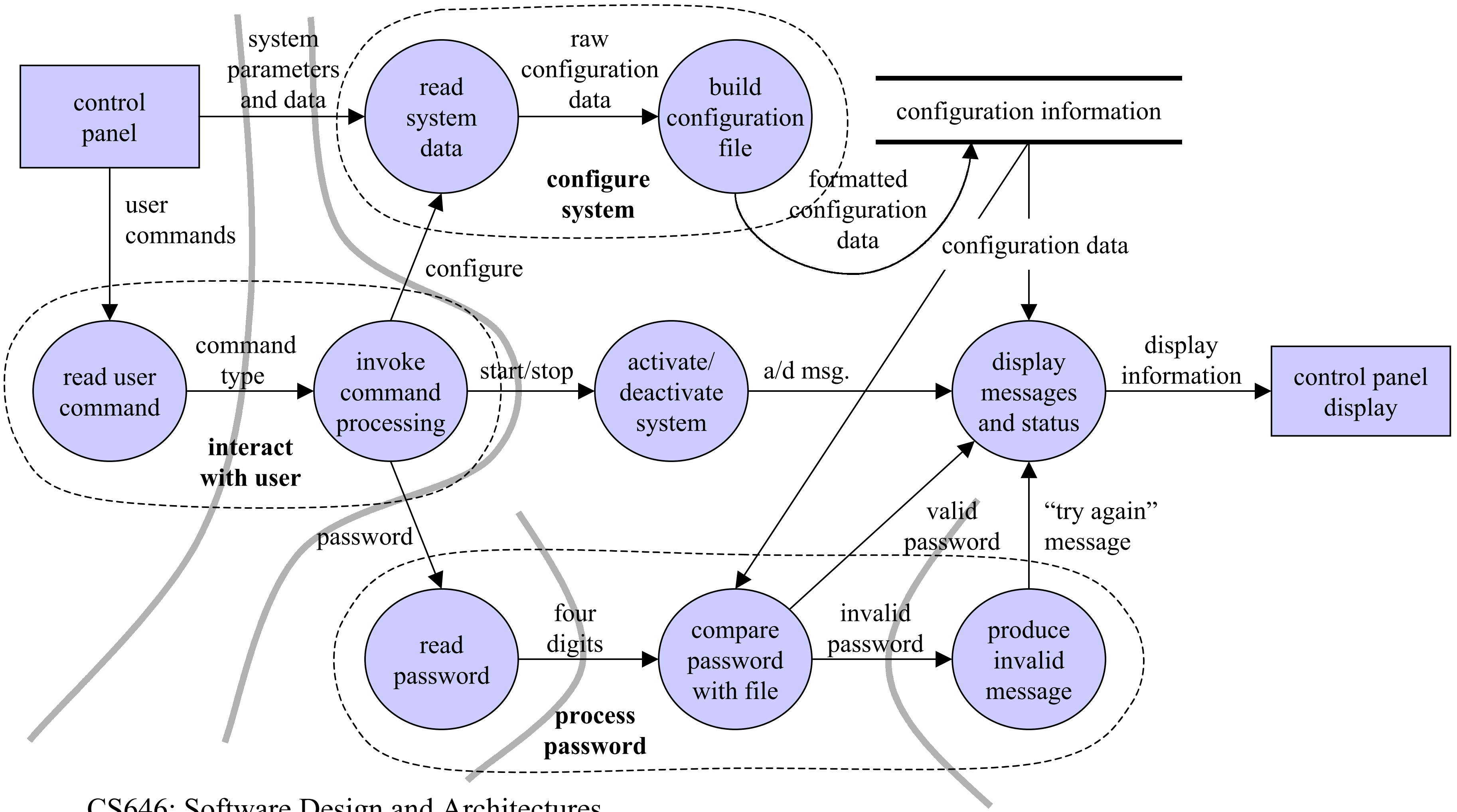


E.g. (cont’d) refined program structure (*monitor sensors*)

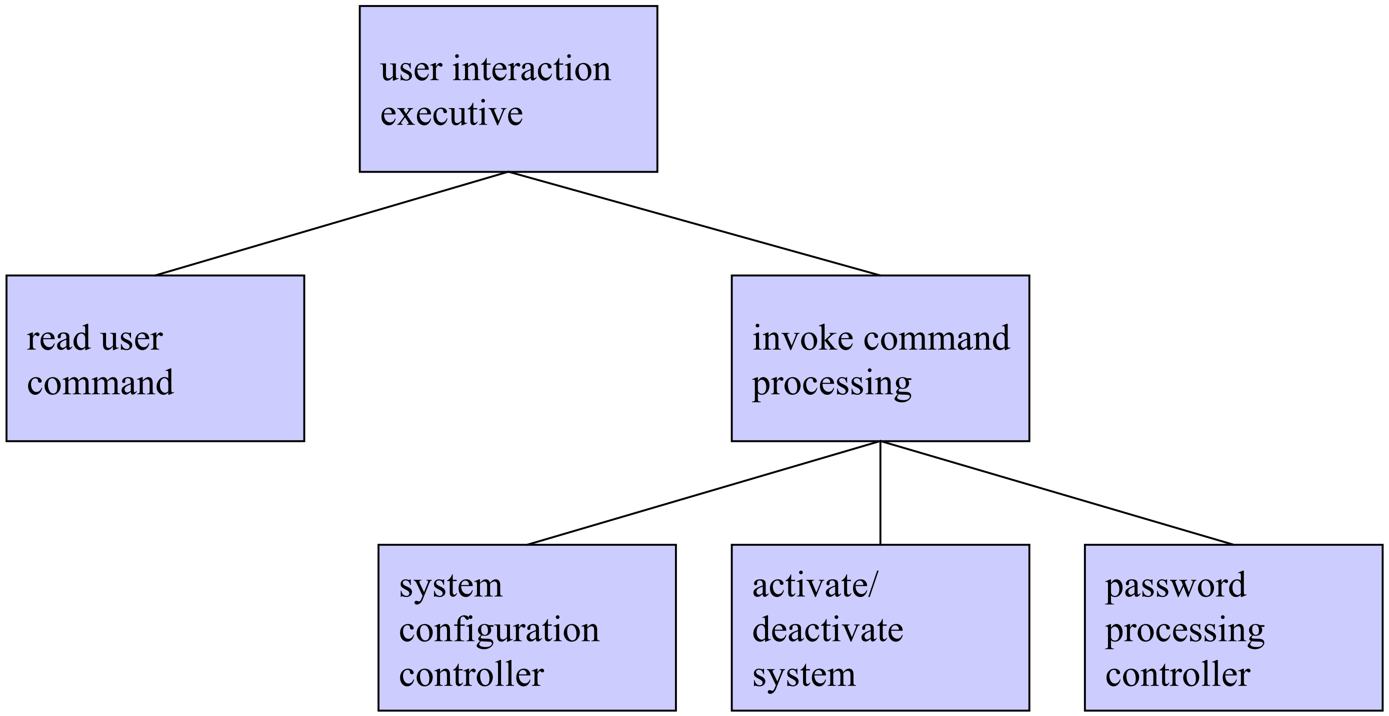


M1 monitor sensors executive
M2 acquire response info
M4 establish alarm conditions
M5 alarm output controller
M3 read sensors
M6 produce display
M7 generate alarm signal
M8 set up connection to phone net
M9 generate pulses to line

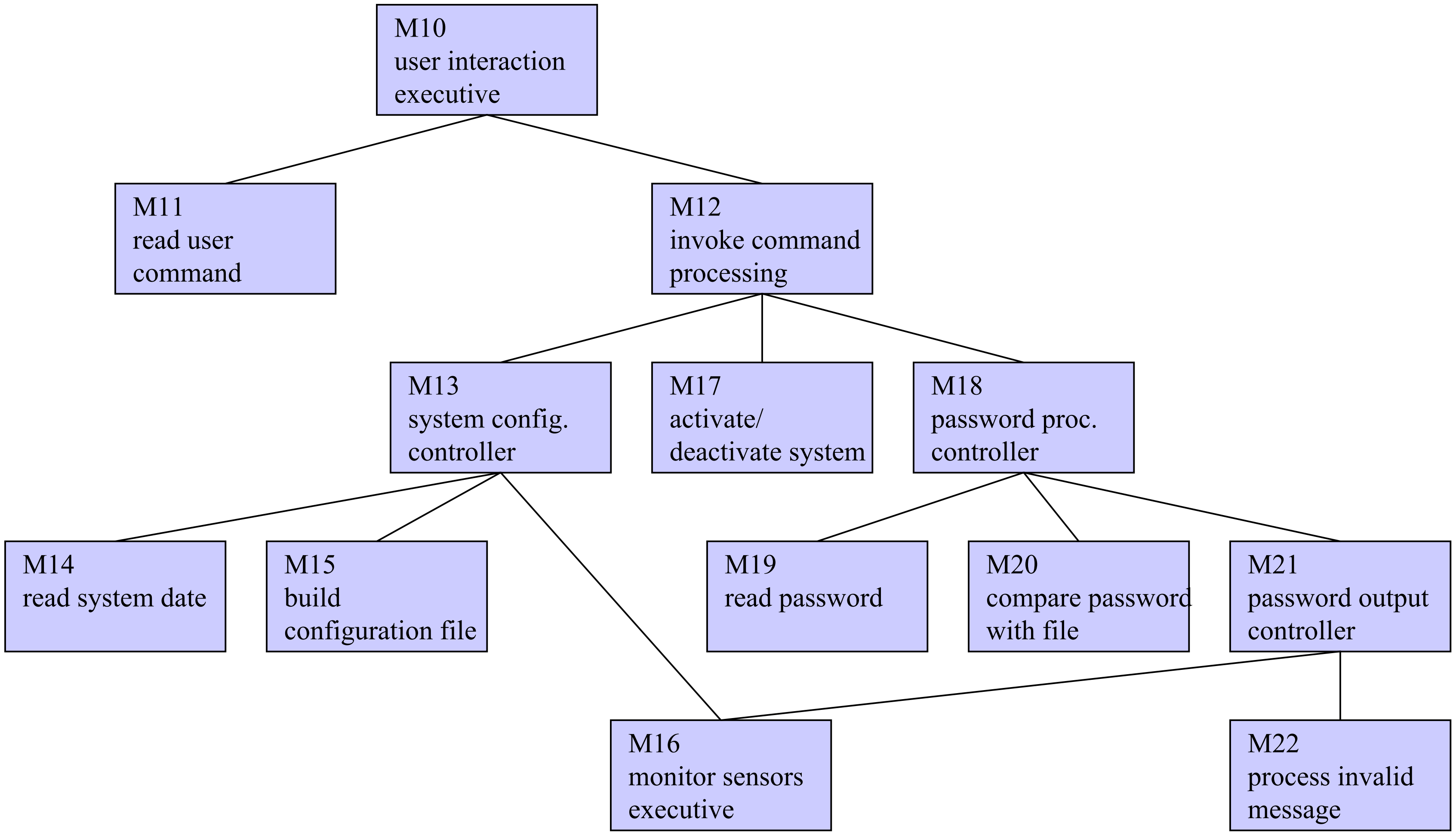
E.g. (cont'd): Level 2 (*user interaction*)



E.g. (cont'd) 1st-level factoring (*user interaction*)


user interaction executive
read user command
invoke command processing
system configuration controller
activate/ deactivate system
password processing controller

E.g. (cont'd) 1st-cut program structure (*user interaction*)


M10
user interaction executive
M11
read user command
M12
invoke command processing
M13
system config. controller
M17
activate/ deactivate system
M18
password proc. controller
M14
read system date
M15
build configuration file
M19
read password
M20
compare password with file
M21
password output controller
M16
monitor sensors executive
M22
process invalid message